Memetic algorithm with an effi cient split procedure for the Team Orienteering Problem with Time Windows

Rym Nesrine Guibadj, Aziz Moukrim

Memetic algorithm with an efficient split procedure for the Team Orienteering Problem with Time Windows

Rym Nesrine GUIBADJ[1] and Aziz MOUKRIM[1]

Université de Technologie de Compiègne
Laboratoire Heudiasyc, UMR 7253 CNRS, 60205 Compiègne, France
{rym-nesrine.guibadj,aziz.moukrim}@hds.utc.fr

Abstract. The Team Orienteering Problem (TOP) is a variant of the vehicle routing problem. Given a set of vertices, each one associated with a score, the goal of TOP is to maximize the sum of the scores collected by a fixed number of vehicles within a certain prescribed time limit. More particularly, the Team Orienteering Problem with Time Windows (TOPTW) imposes the period of time of customer availability as a constraint to assimilate the real world situations. In this paper, we present a memetic algorithm for TOPTW based on the application of split strategy to evaluate an individual. The effectiveness of the proposed MA is shown by many experiments conducted on benchmark problem instances available in the literature. The computational results indicate that the proposed algorithm competes with the heuristic approaches present in the literature and improves best known solutions in 101 instances.

1 Introduction

The Orienteering Problem (OP) was firstly introduced by Tsiligirides [24]. The roots of this problem trace back to the pioneering work of Golden et al. [7] who proved that the OP is NP-hard and used it to formulate and solve the home fuel delivery problem. The name "Orienteering Problem" originates from the sport game of orienteering described in [3]. Later, a new variant of the problem called Team Orienteering Problem (TOP) was introduced since it is widely seen in many real life situations, like for example the routing of technicians [21] and fuel delivery problems [7]. Many heuristics have been successfully applied to TOP. There are four methods that can be considered as the state-of-the-art algorithms in the literature: a variable neighborhood search proposed by Archetti et al. [1], a memetic algorithm [2], a path relinking approach [20] and a PSO-based memetic algorithm [5]. The survey of Vansteenwegen et al. [25] gives a review of the most important contributions on the orienteering literature.

Recently, the Orienteering Problem with Time Windows (OPTW) and the Team Orienteering Problem with Time Windows (TOPTW) have been the interest of many researchers. They are considered as the generalization of OP and TOP with the additional time constraints. In these problems, the service of a

customer must be started within a time window $[e_i, l_i]$ defined by customer i. The vehicle cannot arrive earlier than time e_i and no later than time l_i. A vehicle arriving earlier than the earliest service time of a customer will incur waiting time. The first who considered the time windows in the OP were Kantor and Rosenwein [8]. They solved the problem with a tree heuristic that was more efficient than the classical insertion heuristics. The only exact method that we found was developed by Righini and Salani [17]. The computational time required by their method to solve large problem instances was very expensive. Therefore, most of the researchers focus on developing approximate methods. Montemanni and Gambardella [12] used ant colony optimization to solve the problem, while Vansteenwegen et al. [26] present an iterated local search metaheuristic. In this method, an insert step is combined with a shake step to explore the search space more efficiently. Tricoire et al. [23] defined the Multi-Period Orienteering Problem with Multiple Time Windows (MuPOPTW) as a new problem for scheduling the customer visits of sales representatives. The MuPOPTW is a generalization of OPTW and TOPTW, where customers may be visited on different days, and may have several time windows for each given day. They propose an exact algorithm embedded in a variable neighborhood search method and provide experimental results for their method on standard benchmark of OPTW and TOPTW instances. Lin and Yu [11] presented a simulated annealing based heuristic approach to solve TOPTW. The method proposed by Labadie et al. [10] combines greedy randomized adaptive search procedure (GRASP) with evolutionary local search (ELS). ELS generates multiple distinct child solutions that are further improved by a local search procedure, while GRASP provides multiple starting solutions to ELS. Labadie et al. [9] introduced granular variant to a VNS algorithm in order to improve its efficiency. Firstly, each arc is evaluated with new cost taken into account traveling times, waiting times and profits. Then, an assignment problem is optimally solved and intervals of granularity are created. These intervals determine subset of promising arcs which will be considered during the node sequences construction in the local search procedure.

In this paper, a metaheuristic-based memetic algorithm (MA) is presented for TOPTW. The proposed MA works with permutation encoding and uses an adapted procedure to optimally split a sequence into a set of routes. The rest of the article is organized as follows. The next section is devoted to the formulation of TOPTW. Section 3 presents the detailed description of the proposed method including the solution representation, the optimal split procedure, and other components and parameters. In Section 4, the effectiveness of the proposed algorithm is demonstrated by many computational results based on some benchmark problems. The conclusions are discussed in the final Section 5.

2 Formulation of the problem

TOPTW is modeled with a graph $G = (V, E)$, $V = \{0, 1, 2, ..., n\}$ is the set of vertices where $i \neq 0$ represents a customer and 0 represents the depot. $E = \{(i, j) : i \neq j, i, j \in V\}$ is the edge set. Each vertex $i \in V, i \neq 0$ is associated

with a profit P_i and a service time T_i. The visit of a vertex i can start only within a predefined time window $[e_i, l_i]$. The vehicle v cannot arrive later than the time l_i and if it arrives earlier than e_i, it must wait W_i^v before the service can start. Each edge $(i, j) \in E$ is associated with a travel cost $c_{i,j}$ which is assumed to be symmetric and satisfying the triangle inequality. A *tour* R is represented as an ordered list of q customers from V, so $R = (R[1], \ldots, R[q])$. Each tour begins and ends at the depot vertex. We denote the total profit collected from a tour R as $P(R) = \sum_{i=1}^{i=q} P_{R[i]}$, and the total travel cost or duration $C(R) = c_{0,R[1]} + \sum_{i=1}^{i=q-1} c_{R[i],R[i+1]} + \sum_{i=1}^{i=q} W_{R[i]}^R + \sum_{i=1}^{i=q} T_{R[i]} + c_{R[q],0}$. A tour R is feasible if $C(R) \leq l_0$, l_0 being a latest possible arrival time to the depot, and if each customer is serviced within its time window. The fleet is composed of m identical vehicles. A *solution* S is consequently a set of m (or fewer) feasible tours R in which each customer is visited at most once. The goal is to find a solution S such that $\sum_{R \in S} P(R)$ is maximized. For mixed integer linear programming formulations of TOPTW see [12] and [26].

3 Memetic algorithm

Memetic algorithm is a combination of an evolutionary algorithm and local search framework [13]. The basic idea behind memetic approaches is to combine the advantages of the crossover that discovers unexplored promising regions of the search space, and local optimization that finds good solutions by concentrating the search around these regions. The proposed memetic algorithm is based on permutation encoding. The key feature of the proposed method is the split procedure that allows a reduction of the solution space exploration within the global optimization. We introduce an interesting way to represent solutions of TOPTW, known as giant tours. Each giant tour is in fact a neighborhood of solutions in the search space from which the optimal associated solution can be easily extracted by an evaluation process. Therefore, a heuristic using this representation explores a smaller solution space without any loss of information and has a better chance to reach the global optimum. The good results obtained on several extensions of the routing problems have raised a growing attention on the split strategy [16]. Next, all the details of MA implementation are presented.

3.1 Chromosome and evaluation

The representation of our chromosome consists of an ordered list of all accessible customers in V called a *giant tour*. The giant tour is a permutation of n positive integers, such that each integer corresponds to a customer without trip delimiters. We try to extract m tours from the giant tour while respecting the order of the customers in the sequence. A tour from a permutation π is identified by its starting point i in the sequence and the number of customers following i. A chromosome is evaluated using a tour splitting procedure which optimally partitions π into feasible routes. Using this strategy, the MA searches the set of possible giant tours to find one that gives an optimal solution after splitting.

Bouly et al. [2] proposed an optimal splitting procedure which is specific to TOP. In their method, only tours of maximum length are considered. This means that all customers following i in the sequence are included in the tour as long as all constraints are satisfied, or until the end of the sequence is reached. Such a tour is called *saturated* tour. They proved that solutions containing only saturated tours are dominant. Therefore, only saturated tours were considered in their procedure. Later, Dang et al. [6] introduced a new evaluation procedure in which the limited number of saturated tours is exploited more efficiently to reduce the complexity of the evaluation process. Before reviewing the main idea of the split procedure, we recall the definition of an interval graph [22] as follows. A graph $G = (V, E)$ is an interval graph if there is a mapping I between the vertex set of G and a collection of intervals in the real line such that two vertices of G are adjacent if their respective intervals intersect. Then, for all i and j of V, $[i, j] \in E$ if and only if $I(i) \cap I(j) \neq \emptyset$.

We have extended the split procedure for TOPTW to tackle time windows. When defining a saturated tour R starting with x, we should make sure that each customer is served within its time window and that $C(R) \leq l_0$, where l_0 is the latest possible arrival time to the depot and $C(R)$ the total travel duration. So, a waiting time is added each time the vehicle arrives at a customer before the beginning of his time window. The set of extracted tours from a giant tour can be mapped to the set of vertices of an interval graph X. An edge in X indicates the presence of shared customers between the associated tours. A split procedure looks for m tours without any shared customer such that the sum of their profit is maximized. So this is equivalent to solve a knapsack problem with the conflict graph X. In this particular knapsack problem, the number of items is equal to the number of possible tours. This number is equal to n when only saturated tours are considered. The weight of each item is one and the capacity of the knapsack is m. Such a problem can be solved in $O(m \cdot n)$ time and space [18].

Proposition 1 *Given a TOPTW instance where m is the maximum number of available vehicles and π a permutation of n customers, the split procedure of π can be done optimally in $O(m \cdot n)$ time and space.*

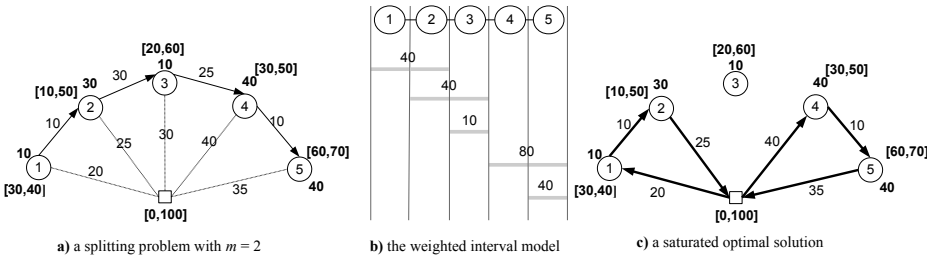

Fig. 1. An example of splitting problem.

The evaluation process is performed with dynamic programming technique: A two-dimensional array of size $m \cdot n$ is used to memorize the maximum reachable profit during process. Then, a backtrack is performed in order to determine the tours corresponding to the optimal solution. The first graph of Figure 1 shows a sequence $S = (1, 2, 3, 4, 5)$ where each customer has a profit and a time window given in the square brackets. To simplify, we assume that the service times are set to 0, the number of the vehicles m used is equal to 2, and the maximum operation time l_0 is 100. The interval model is given in the Figure 1.b. The first interval $[1, 2]$ for example with weight 40 corresponds to the collected profit of the trip $(d, 1, 2, a)$. Vehicle leaves the depot at time 0, waits 10 units of time at node 1 before leaving it to serve node 2 at time 40. The customer 3 cannot be included in the trip, since its time window is already closed when the vehicle reaches it at time 70. The other intervals $[i, j]$ of the graph are similarly defined. The maximum score obtained in the solving steps is equal to 120. Finally, we give the optimal solution obtained by the algorithm in Figure 1.c. It is composed of two tours starting respectively with customers 1 and 4.

3.2 Population

A small part of the initial population is created with a fast heuristic procedure and the remainder is generated randomly. In the proposed Iterative Destruction/Construction Heuristic (IDCH), we build a feasible solution by inserting at every iteration an unrouted customer. This process is performed using Best Insertion Algorithm (BIA). Initially, IDCH removes a limited random number of customers $D \in \{1, 2, 3\}$ from the current solution. Then, the travel cost of tours is reduced using 2-opt* and Or-opt exchanges [14]. In the next step, we rebuild the solution by re-inserting unrouted customers in all possible ways. To ensure that the feasibility of an insertion is verified in $O(1)$, we record for each already included customer i in a route r, its waiting time W_i^r and the maximum delay allowed for the service $Maxshift_i^r$. All feasible insertions of each unserved customer u between two couple of adjacent customers i and j are evaluated. This is done according to a suitable cost function $f(u) = Shift_u/(P_u)^\alpha$ where $Shift_u = (c_{i,u} + W_i^r + T_u + c_{u,j} - c_{i,j})$. The feasible insertion that minimizes the cost is then processed. In addition, priority coefficient $prio_u$ is associated to each customer u. Whenever the customer is not routed through a construction phase its priority is increased by the value of its profit. The customer u that has a lager $prio_u$ is more likely to be inserted. When a limited number of iterations $iter_{perturb} = n$ is reached without a strict improvement, a method of diversification is performed. Diversification stands for random moves that can deteriorate the current solution by removing a large number of customers $D_{perturb} \in [1, n/m]$. The destruction and construction phases are iterated until $iter_{max} = n^2$ iterations without improvement.

3.3 Selection and crossover

In this work, the binary tournament method [15] is adopted to select a couple of parents among the population. Two chromosomes are randomly selected in the population, and the chromosome with the best evaluation becomes the first parent. The tournament is repeated for the second parent. These parents are then combined using linear order crossover or LOX [15]. LOX first chooses two cut points randomly and passes the section enclosed by the cut points from one parent to child. Then, the unpassed customers are placed in the unfilled positions using the order of their occurrence in the other parent.

3.4 Local search engine

When a new child is computed with the crossover operator, the local search scheme is applied with a probability *pm*. Neighborhoods are selected in a random order. The search in a given neighborhood is stopped as soon as a better solution is found. Then, a new neighborhood is chosen randomly. This process is stopped when all neighborhoods fail to bring out an improvement to the current solution. The set of local search operators used in the Memetic Algorithm are:

- *2-opt* operator*: two routes r_1 and r_2 are divided into two parts. Then the first part of r_1 is connected to the second part of r_2, while the first part of r_2 is connected to the second part of r_1.
- *Or-opt operator*: consider a sequence of one, two or three consecutive customers in the the current solution, and move the sequence to another location in the same route.
- *destruction/repair operator*: first, a random number of customers (between 1 and $\frac{n}{m}$) is removed from an identified solution. Then, the lowest possible insertion cost $\frac{Shift_i}{(P_i)^\alpha}$ of each unrouted customer i is evaluated. The visit with the lowest ratio will be selected for insertion.
- *shift operator*: a customer is removed from its current position and is relocated at another one. Every possible insertion position for every customer is considered.
- *swap operator*: positions of every two customers in the sequence are exchanged.

3.5 Population update

When an offspring solution s_{new} is created by the crossover operator presented in Section 3.3 and improved by the local search algorithm described in Section 3.4, we decide if the improved offspring should be inserted into the population and which existing solution of the population should be replaced. Basically, our decision is made based on both: the solution quality and the distance between solutions in the population. The update procedure is applied if the performance of new solution s_{new} is better than the worst individual. Population is a list of solutions sorted in descending order according to two criteria: the total collected

profit and the travel cost/time. Two solutions are said to be similar or identical if the evaluation procedure returns the same profit and a difference in travel cost/time lower than a value δ. If there is a solution s similar to s_{new}, then s is replaced with s_{new}. Otherwise the worst individual is deleted and the new solution is inserted into the population.

3.6 Basic algorithm

The proposed Memetic Algorithm associates all the elements described above. Algorithm 1 presents a synthetic view of the whole process. The algorithm starts

Algorithm 1: Basic algorithm

```
   Data: POP a population of N solutions;
   Result: S_POP[N − 1] best solution found;
 1 begin
 2     initialize and evaluate each solution in POP (see Section 3.2);
 3     while NOT(stopping condition) do
 4         select 2 parents POP[p1] and POP[p2] using binary tournament;
 5         C ← LOX(POP[p1], POP[p2]) ;
 6         if rand(0, 1) < pm then
 7             apply local search on C (see Section 3.4);
 8         if f(C) ≥ f(POP[0])(see Section 3.5) then
 9             if ∄p||(f(POP[p]) = f(C)) then
10                 eject POP[0] from POP ;
11                 reset stopping condition ;
12             else
13                 update stopping condition;
14             insert or replace C in right place in POP ;
15         else
16             update stopping condition;
```

with an initial set of solutions, called population. During each iteration, two parents are selected and crossover operator is applied to create a new solution. The obtained child chromosome has a probability pm of being mutated using a set of local search techniques repeatedly. Finally, it is inserted within the population according to its fitness evaluation. The stopping criterion for MA is after reaching a maximum number of iterations without improvement. That is to say after reaching the number of iterations where the child chromosome has the same fitness as an existing chromosome in the population, or when its evaluation is worse than the worst chromosome in the current population.

4 Numerical results

We used 56 instances designed by Solomon [19] and 20 instances designed by Cordeau et al. [4] to test our new proposed algorithm. Solomon's 100-customer instances are divided into *random*, *clustered* and *randomclustered* categories. In Cordeau's instances, the number of customers varies between 48 and 288. A third set of benchmark was introduced by Vansteenwegen et al. [26] using the original instances of Solomon and Cordeau. In these instances, the number of vehicles considered allows to visit all customers that is why the optimal solutions of these instances are known since they are equal to the sum of customers' profits. Travel time between two customers is assumed to be equal to the travel distance. It is rounded down to the first decimal for the Solomon's instances and to the second decimal for the Cordeau's instances. The whole algorithmic approach was implemented in C++ using the Standard Template Library (STL) for data structures and was compiled using the GNU GCC compiler on an AMD Opteron 2.60 GHz in a Linux environment.

4.1 Parameter setting

A number of different alternative values were tested and the ones selected are those that gave the best computational results concerning both the quality of the solution and the computational time needed to achieve this solution. When the population is initialized, 5 chromosomes are generated by the IDCH heuristic and the rest (35) are generated randomly. The similarity measurement of individuals δ is set to 0.01 and the local search rate pm is calculated as: $1 - \frac{iter}{itermax}$ where $iter$ is the number of consecutive iterations without improvement. The algorithm stops when $iter$ reaches $itermax = k * n/m$. The cost function $C(u) = Shift_u/(P_u)^{\alpha}$ of the BIA heuristic uses a random value of α generated in $[1, 3]$. This control parameter makes our IDCH less predictable and actually a randomized heuristic. Moreover, the score becomes more relevant than the time consumption when deciding which unrouted client is the most promising to insert. If α is set to 1, the obtained results are worse. Finally only two parameters are required to be tuned, they are the stopping condition k and the population size N. Computational experiments were conducted on a representative subset of the problem characteristics (problem size, distribution of customer location, and time windows characteristic). This small subset includes 40 instances: 6 problems from Solomon's instances and 4 problems from Cordeau's instances with $m = 1, 2, 3, 4$. The value of k and N were varied from 10 up to 50 with steps of 10. This results 25 different (k, N) settings to be tested. The algorithm was run 5 times on different randomly generated seeds for each instance. For an overall performance comparison between different configurations, we use two following measures. The first one is the relative gap to the best known solutions, denoted $rpe(\%)$ and the second is the average computational time in seconds CPU_{avg}. The results for each of the 25 parameter combinations tested are illustrated in Figure 2. We adopt the parameter settings $(10, 40)$ which gives a good trade off between algorithm performance and computational time.

Instance Set	ACS		ILS		VNS		GRASP-ELS		SA		GVNS		MA	
	rpe	cpu_{avg}	rpe	cpu_{avg}	rpe	cpu_{avg}	rpe	cpu_{avg}	rpe	cpu_{avg}	rpe	cpu_{avg}	rpe	cpu_{avg}
m=1														
c100	**0**	6.34	1.11	0.33	**0**	98.39	**0**	22.59	**0**	21.07	0.56	166.46	**0**	0.98
r100	**0**	383.40	1.90	0.19	**0**	89.10	0.11	3.51	0.11	23.34	1.72	29.43	**0**	5.38
rc100	**0**	143.21	2.92	0.23	**0**	65.21	0.33	1.99	**0**	22.19	1.88	9.80	**0**	1.59
c200	0.40	342.61	2.28	1.71	**0**	560.17	0.40	32.18	0.13	37.49	0.55	192.40	**0**	122.40
r200	2.18	1556.70	2.89	1.66	0.40	1065.82	0.59	11.18	1.29	45.83	2.44	33.82	**-0.52**	236.10
rc200	1.23	1544.55	3.43	1.63	0.07	869.41	1.37	8.21	0.96	50.25	2.53	16.01	**-0.02**	201.52
pr01-pr10	1.05	1626.61	4.72	1.75	**0**	822.07	0.73	5.03	0.97	112.21	0.54	12.37	**-0.02**	485.98
pr11-pr20	10.73	887.66	9.11	1.98	0.93	1045.93	1.70	7.90	3.25	162.40	2.71	24.22	**0.39**	903.08
m=2														
c100	0.15	818.00	0.94	1.08	**0**	87.98	**0**	70.94	**0**	26.42	0.47	139.53	**0**	70.09
r100	0.34	1559.36	2.27	0.87	0.06	63.46	0.92	7.97	0.14	36.63	1.10	60.34	**-0.12**	45.98
rc100	0.38	1375.78	2.47	0.71	0.23	55.16	1.46	4.66	0.19	40.48	0.78	20.31	**0**	46.33
c200	1.27	1398.10	2.54	3.46	0.51	545.65	0.09	29.26	1.18	53.66	0.25	33.79	**0**	164.93
r200	3.11	2735.15	2.69	2.27	0.20	1015.08	0.28	17.58	0.53	91.40	0.62	14.73	**-0.57**	634.67
rc200	2.64	2342.72	4.08	2.20	0.43	804.83	0.59	17.14	1.18	80.10	1.62	12.76	**-0.60**	355.97
pr01-pr10	2.35	1889.66	5.99	4.76	0.63	524.83	0.87	19.46	2.21	173.93	0.57	39.09	**-0.44**	1291.54
pr11-pr20	4.79	2384.81	7.65	5.21	1.04	618.78	2.21	28.77	3.66	201.63	0.98	82.44	**-0.24**	2144.27
m=3														
c100	0.11	1043.24	2.44	1.50	**0**	85.49	0.13	86.74	0.22	35.26	0.34	165.01	**0**	70.77
r100	0.55	1668.86	1.78	1.67	0.21	61.91	0.89	13.86	0.38	56.07	1.21	73.93	**-0.01**	58.56
rc100	1.19	1476.81	3.14	1.11	0.36	60.62	1.83	8.65	0.64	42.80	0.91	33.68	**-0.01**	54.72
c200	0.55	1413.11	1.98	2.08	0.16	196.80	0.45	26.75	0.35	53.93	0.64	55.42	**-0.10**	104.73
r200	0.13	1171.65	0.30	1.36	0.03	321.65	**0**	2.49	0.08	41.95	0.11	6.97	**0**	**74.22**
rc200	0.37	1607.85	1.37	1.73	0.04	404.01	0.06	8.34	0.20	58.98	0.25	7.41	**-0.07**	212.43
pr01-pr10	3.01	2163.80	6.57	9.24	1.50	473.20	1.31	40.55	2.33	197.01	0.35	85.90	**-0.33**	1416.21
pr11-pr20	5.19	2383.29	8.91	9.69	1.48	517.48	2.00	42.95	3.51	251.83	0.72	150.73	**-0.71**	2388.19
m=4														
c100	0.47	1056.05	2.93	2.57	0.09	81.87	0.50	84.58	0.36	49.51	0.85	133.22	**-0.19**	106.15
r100	0.99	1652.54	3.25	2.60	0.24	61.17	0.88	24.18	0.67	58.38	1.15	84.74	**-0.11**	79.46
rc100	0.92	1854.00	3.07	1.98	0.34	58.47	1.43	13.35	0.26	68.13	0.85	36.91	**-0.24**	57.66
c200	**0**	7.70	**0**	1.00	**0**	104.78	**0**	0.01	**0**	41.76	**0**	0.55	**0**	0.04
r200	**0**	126.46	**0**	0.87	**0**	150.74	**0**	0.03	**0**	39.71	**0**	0.27	**0**	0.10
rc200	**0**	646.72	**0**	1.24	**0**	164.56	**0**	0.03	**0**	40.15	**0**	0.88	**0**	0.15
pr01-pr10	2.34	2447.70	6.63	14.07	1.40	403.17	1.42	45.75	1.76	255.57	0.60	127.33	**-1.12**	1807.40
pr11-pr20	4.18	2583.50	7.16	13.74	0.90	408.01	1.20	65.33	2.57	283.98	0.64	232.64	**-2.23**	2784.70
Average	1.65	1401.79	3.38	3.09	0.36	375.62	0.74	22.60	0.96	88.30	0.87	64.34	**-0.23**	524.00

Table 1. Performance comparison based on RPE average for each data set of the standard benchmark.

Instance Set	ILS		GRASP-ELS		SA		GVNS		MA	
	$arpe$	cpu_{avg}	$arpe$	cpu_{avg}	$arpe$	cpu_{avg}	$arpe$	cpu_{avg}	$arpe$	cpu_{avg}
new Solomon's instances	1.12	2.38	0.35	70.34	0.30	35.70	0.65	16.92	**0.04**	43.02
new Cordeau's instances	2.32	30.41	1.04	565.98	0.92	71.48	1.25	51.34	**0.76**	112.63
Average	1.72	16.40	0.70	318.16	0.61	53.59	0.95	34.13	**0.40**	77.82

Table 2. Performance comparison based on ARPE average for each data set of the new benchmark.

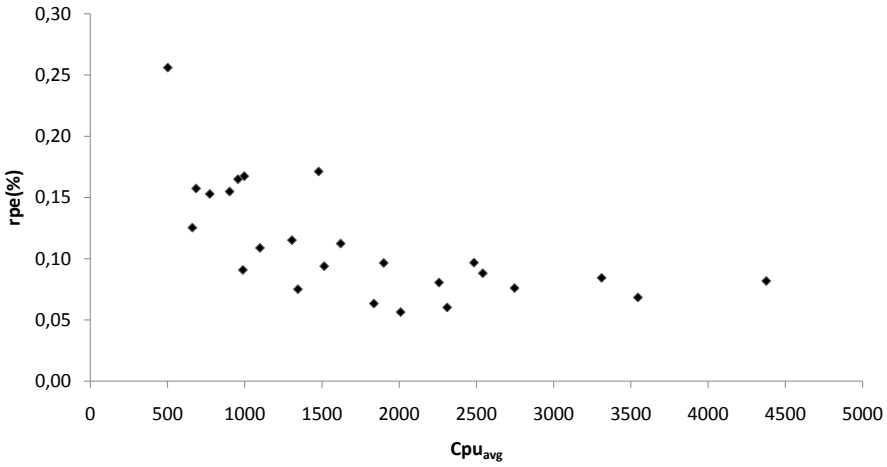

Fig. 2. Pareto front solutions obtained with different settings of the stopping condition k and the population size N

4.2 Performance comparison

In order to investigate the performance of the proposed MA for TOPTW, we compare it with: the Ant Colony System (ACS) of [12], the Iterated Local Search (ILS) of [26], the Variable Neighborhood Search (VNS) of [23], the Simulated Annealing approach (SA) of [11], the Greedy Randomized Adaptive Search procedure of (GRASP-ELS) [10] and the Granular Variable Neighborhood Search (GVNS) of [9]. The results of GVNS, GRASP-ELS and ACS were obtained with 5 runs of the algorithm on each instance. VNS was run 10 times per instance while ILS and SA were executed only once. We used the same protocol as in the state-of-the-art methods and run MA 5 times for each instance. The quality of the produced solutions is given in terms of the relative percentage error (RPE) for the standard benchmark and in terms of the average relative percentage error (ARPE) for the new data set where there exists a solution visiting all customers. Tables 1 and 2 summarize the comparison and report the percentage error (RPE or $ARPE$) and the average computational time in seconds CPU_{avg} for each instance set. MA produces the best relative gap which is equal to $-0,23\%$ for the standard benchmark and $0,40\%$ for the new data set. The first conclusion that can be drawn from these tables is that MA is very competitive compared to the others methods. It outperforms the other methods and improves 101 instances for which the optimal solution remains unknown. However, one should note that MA is far more time consuming. Actually, on the largest instances. MA needs more time to get good quality solutions. The reason appears to be that a lot of time is consumed by local-search operators. This is necessary to take entirely advantage of the MA component.

5 Conclusion

In this paper, a Memetic Algorithm was proposed for the Team Orienteering Problem with Time Windows. The key feature of our algorithm is the use of an Optimal Split procedure especially intended for TOPTW that runs in $O(m \cdot n)$. The proposed algorithm integrates several optimization methods, including heuristic approaches, a crossover operator, a local search optimization procedure and a quality-and-diversity based population updating strategy. The computational results obtained prove the efficiency of our memetic algorithm for TOPTW in comparison with the existing ones. The algorithm brings further improvements and has allowed the identification of new best known solutions. The method is also very flexible in the sense that it can address many problem variants with a unified methodology and common parameter settings. Future work will focus on extending the methodology to a wider array of vehicle routing problems with time windows.

Bibliography

[1] Archetti, C., Hertz, A., Speranza, M.G.: Metaheuristics for the team orienteering problem. Journal of Heuristics 13(1) (February 2007)

[2] Bouly, H., Dang, D.C., Moukrim, A.: A memetic algorithm for the team orienteering problem. 4OR 8(1), 49–70 (2010)

[3] Chao, I.M., Golden, B., Wasil, E.A.: The team orienteering problem. European Journal of Operational Research 88, 464–474 (1996)

[4] Cordeau, J.F., Gendreau, M., Laporte, G.: A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks 30(2), 105–119 (1997)

[5] Dang, D.C., Guibadj, R.N., Moukrim, A.: A pso-based memetic algorithm for the team orienteering problem. In: EvoApplications. pp. 471–480 (2011)

[6] Dang, D.C., Guibadj, R.N., Moukrim, A.: An effective pso-inspired algorithm for the team orienteering problem. European Journal of Operational Research 229(2), 332 – 344 (2013)

[7] Golden, B., Levy, L., Vohra, R.: The orienteering problem. Naval Research Logistics 34, 307–318 (1987)

[8] Kantor, M.G., Rosenwein, M.B.: The orienteering problem with time windows. The Journal of the Operational Research Society 43(6), 629–635 (1992)

[9] Labadie, N., Mansini, R., Melechovský, J., Calvo, R.W.: The team orienteering problem with time windows: An lp-based granular variable neighborhood search. European Journal of Operational Research 220(1), 15–27 (2012)

[10] Labadie, N., Melechovský, J., Calvo, R.W.: Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. Journal of Heuristics 17(6), 7296–753 (2011)

[11] Lin, S.W., Yu, V.F.: A simulated annealing heuristic for the team orienteering problem with time windows. European Journal of Operational Research 217(1), 94–107 (2012)
[12] Montemanni, R., Gambardella, L.: Ant colony system for team orienteering problems with time windows. Foundations of Computing and Decision Sciences 34(4), 287–306 (2009)
[13] Moscato, P.: New ideas in optimization. chap. Memetic algorithms: a short introduction, pp. 219–234. McGraw-Hill Ltd., UK, Maidenhead, UK, England (1999)
[14] Potvin, J.Y., Kervahut, T., Garcia, B.L., Rousseau, J.M.: The vehicle routing problem with time windows part i: Tabu search. INFORMS Journal on Computing 8(2), 158–164 (1996)
[15] Prins, C.: A simple and effective evolutionary algorithm for the vehicle routing problem. Computers & Operations Research 31(12), 1985–2002 (2004)
[16] Prins, C., Labadie, N., Reghioui, M.: Tour splitting algorithms for vehicle routing problems. International Journal of Production Research 47(2), 507–535 (2009)
[17] Righini, G., Salani, M.: Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. Computers & Operations Research 36(4), 1191–1203 (2009)
[18] Sadykov, R., Vanderbeck, F.: Bin packing with conflicts: a generic branch-and-price algorithm (2012), preprint accepted for publication in INFORMS Journal on Computing
[19] Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research 35(2), 254–265 (1987)
[20] Souffriau, W., Vansteenwegen, P., Vanden Berghe, G., Van Oudheusden, D.: A path relinking approach for the team orienteering problem. Computers & Operations Research 37(11), 1853–1859 (2010)
[21] Tang, H., Miller-Hooks, E.: A tabu search heuristic for the team orienteering problem. Computers & Operations Research 32, 1379–1407 (2005)
[22] Tarjan, R.E.: Graph theory and gaussian elimination. Tech. rep., Stanford University (1975)
[23] Tricoire, F., Romauch, M., Doerner, K.F., Hartl, R.F.: Heuristics for the multi-period orienteering problem with multiple time windows. Computers & Operations Research 37, 351–367 (2010)
[24] Tsiligirides, T.: Heuristic methods applied to orienteering. Journal of the Operational Research Society 35(9), 797–809 (1984)
[25] Vansteenwegen, P., Souffriau, W., Van Oudheusden, D.: The orienteering problem: A survey. European Journal of Operational Research 209(1), 1–10 (2011)
[26] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., Van Oudheusden, D.: Iterated local search for the team orienteering problem with time windows. Computers & Operations Research 36(12), 3281–3290 (2009)